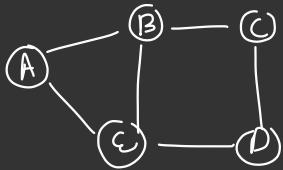


[GRAPHS - II]



Storage

① Edge list

A, B, C, D, E

LL ArrayList
 ↑
 { A B, B C, ... }

③ Adj List [array of list,
hashmap<string, list<string> >]

(A) \rightarrow (B, C)

(B) \rightarrow (C, A)

(C) \rightarrow (B, D)

:

② Adj Mat
A B C D E

	A	B	C	D	E
A	1	1	1	0	0
B	1	1	1	1	0
C	1	1	1	1	1
D	0	1	1	1	0
E	0	0	1	0	1

bool / int

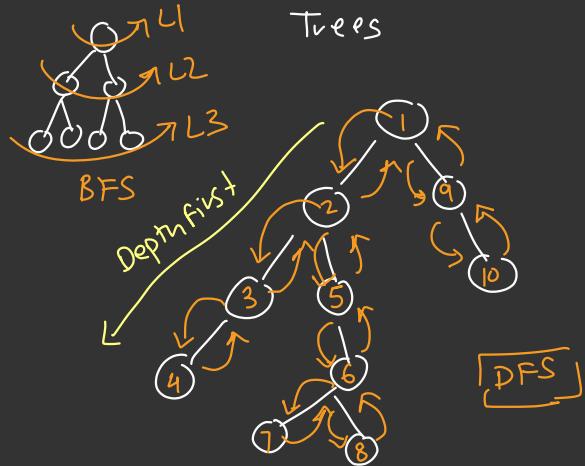
Algo to Traverse the graph

itr \rightarrow ① BFS \rightarrow application: SSSP on unweighted graphs

rec \rightarrow ② DFS

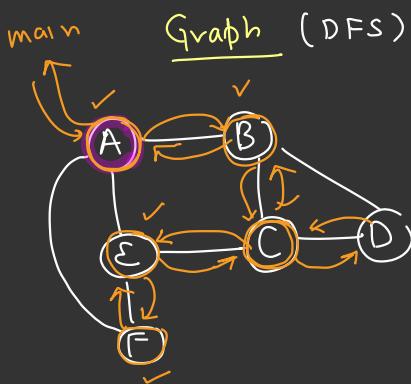
Depth First Search

- Rec way of traversing the graph



Preorder (DFS)

- Root
- left
- Right



adj list

\downarrow

A $\rightarrow (\underline{\text{B}}, \text{E}, \text{F})$

B $\rightarrow (\text{A}, \underline{\text{C}}, \text{D})$

C $\rightarrow (\text{E}, \underline{\text{B}}, \underline{\text{D}})$

D $\rightarrow (\text{B}, \text{C})$

E $\rightarrow (\text{A}, \text{C}, \text{F})$

F $\rightarrow (\text{E}, \text{A})$

Node

\downarrow

A , B , C , E , F , D

Children

- maintain a visited array to stop visiting same node again

wrapper Fn
`dfs (list<int> adjList[], int v) {`

$v=4$
 $\{1, 2, 3\}$

only once
 \Rightarrow `bool visited[] = new bool[v]`
`dfsHelper (adjList, visited, 0)`

// default value is
 false, otherwise
 set it to
 false

logic void
`dfsHelper (list<int> adjList[], bool visited[], int node) {`

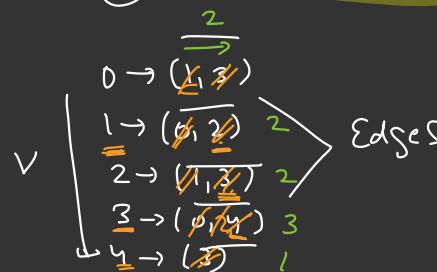
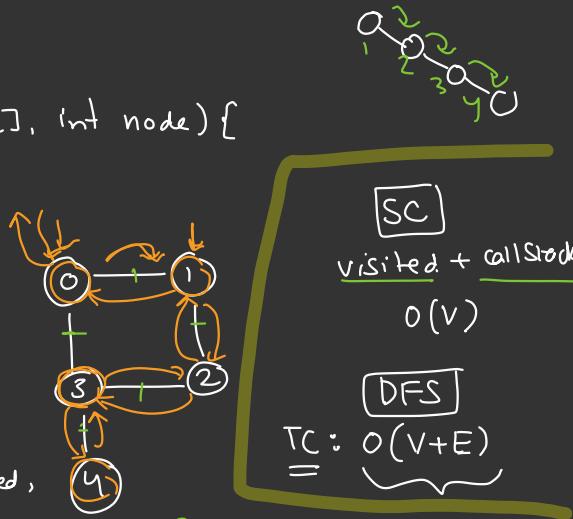
`print(node)`
`visited[node] = true`

0
 called
 once for
 each
 node
 $O(v + 2E)$
 $= O(v + E)$

`for (nbr : adjList[node]) {`
`if (visited[nbr] == false) {`
`dfsHelper (adjList, visited, nbr)`

y

return (optional)



class graph {

list<int> adjList[] → no need to pass as parameter
→ shared across all the fun

=

bfs()

:

dfs()

:

}

list<int> adjList[]

bfs(—, —, →)

dfs(

)

Assuming N Nodes $\{0, 1, \dots, N-1\}$

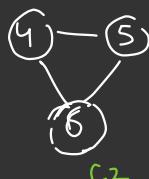
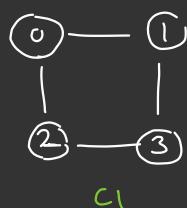
Q. N Nodes

$\{A, B, C, D, E, \dots, X, \text{Delhi}, \dots\}$

if nodes are not sequence
✓
hashmap<string, bool>
visited;

PROBLEM. $N=9$, find no of connected components in the graph.

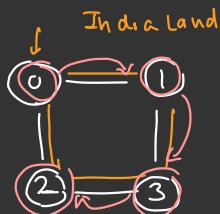
Input \Rightarrow create it
 $O \rightarrow 1, 2$
 $1 \rightarrow 0, 3$
 $2 \rightarrow 0, 3$
 $3 \rightarrow 1, 2$
 $4 \rightarrow 5, 6$
 $5 \rightarrow 4, 6$
 $6 \rightarrow 4, 5$
 $7 \rightarrow 8$



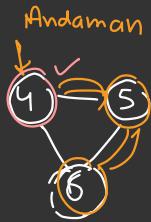
= 3 components

A component is said to be connected if from every node we can visit all the nodes inside the component.

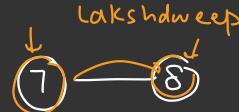
one graph



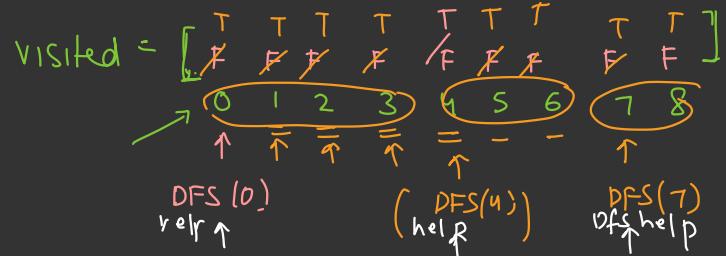
$C_1 = \{0, 1, 2, 3\}$



$C_2 = \{4, 5, 6\}$



$C_3 = \{7, 8\}$



$$C_1 = 4$$

$$C_2 = 3$$

$$C_3 = 2$$

```
list <int>
int dfs ( adjList[], int v ) {
```

connected
component

```
bool visited[] = new bool[v]
count = 0
init(visited, false)
```

```
for( i=0 ; i<v ; i++ ) {
    if( !visited[i] ) {
        => dfs Helper( adjList, visited, i ) → Print all
            nodes
            in
            the
            component
        count++
```

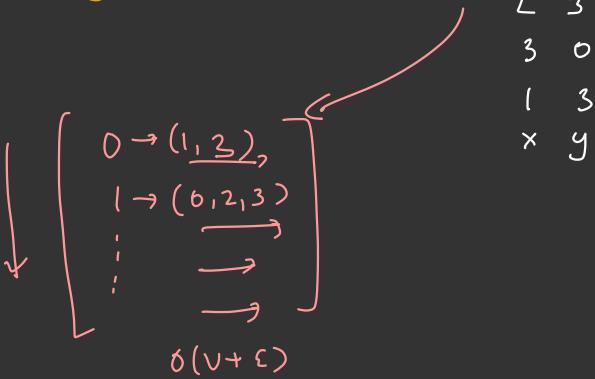
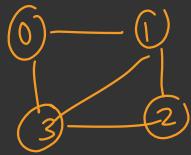
3 return count;

OR it can BFS start from i

TC : $O(V+E)$ for BFS
 SC : $O(V+E)$ > DFS.

Input \rightarrow edge list

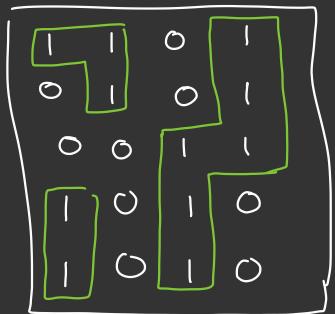
L
 $\underline{\underline{\text{adjList}}}(O(V+E))$



4 5
0 1
1 2
2 3
3 0
1 3
x y

Graph g;
N = input()
E = input()
for (i=0; i < E; i++) {
 Read x, y
 g addEdge(x, y)
 3
 $\Theta(v + \varepsilon)$ Space

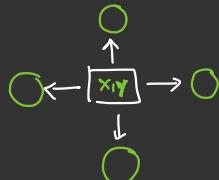
2D Matrix



↓
⇒ 3 islands

0 - water
1 - land

No of islands

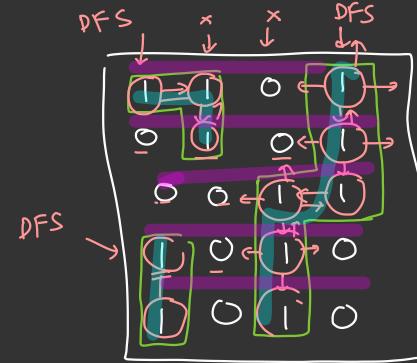
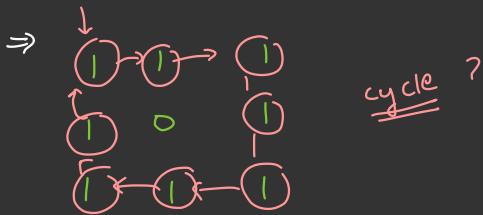
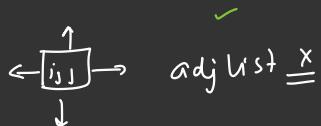


```
int getIslands ( mat[ ][ ] , N, M )
```

TRY it

10²⁰

$\rightarrow \text{O} \Rightarrow \text{nbrs}$



Implicit
Graph

3 calls

dfs Helper (mat[], vis[], int i, int j) {

 if (mat[i][j] == 0 or i, j is outside mat or visited[i][j] == true)

 return

 vis[i][j] = True

 dfs Helper (mat, vis, i, j-1)

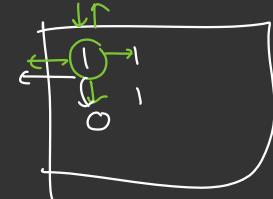
 - dfs Helper (mat, vis, i, j+1)

 - dfs Helper (mat, vis, i+1, j)

 - dfs Helper (mat, vis, i-1, j)

Rec on
2D
array
==

}



```

getIslands → dfs ( mat[], N, M ) {
    bool visited[N][N] = { false } // create and set it false
    count = 0
    for (i = 0 _____ M-1)
        for (j = 0 _____ N-1) {
            if (mat[i][j] == 1 && !visited[i][j]) {
                dfsHelper(mat, vis, i, j)
                count++
            }
        }
    return count,
}

```

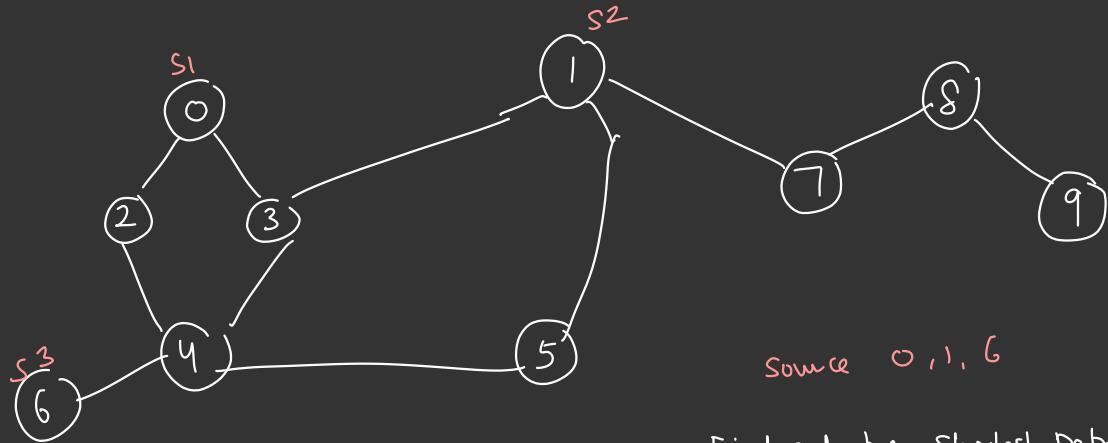


0 1 0 0 -
0 0 1 0 -
0 0 0 0 -

$$\begin{aligned}
 \Rightarrow TC &= O(N^2) \quad O(N^2 + N^2) \\
 SC &= O(N^2)
 \end{aligned}$$

BFS
· DFS

"Multi-source BFS



Source 0, 1, 6

Find out the shortest path len

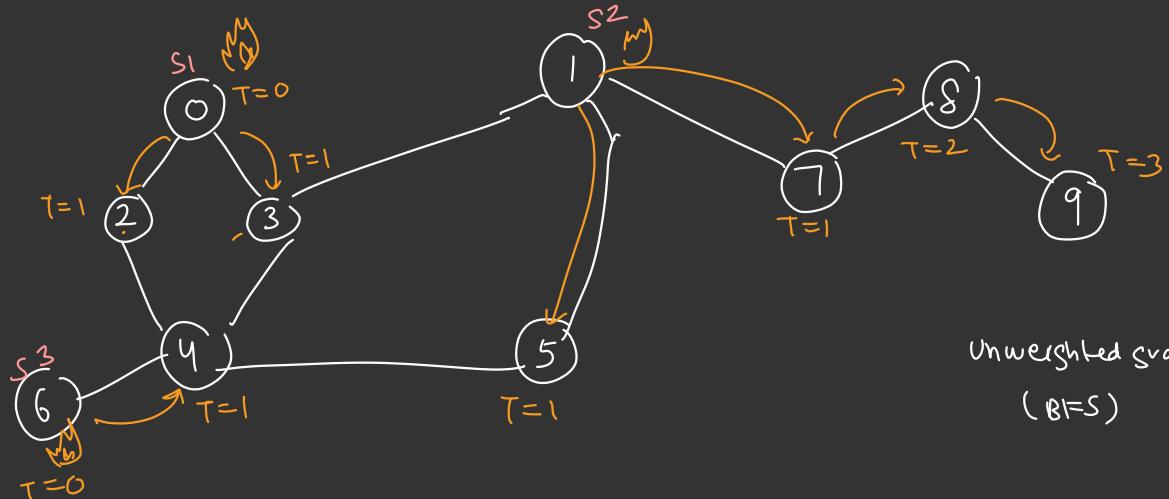
Examples

$$d(8) = 2$$

$$d[4] = 1$$

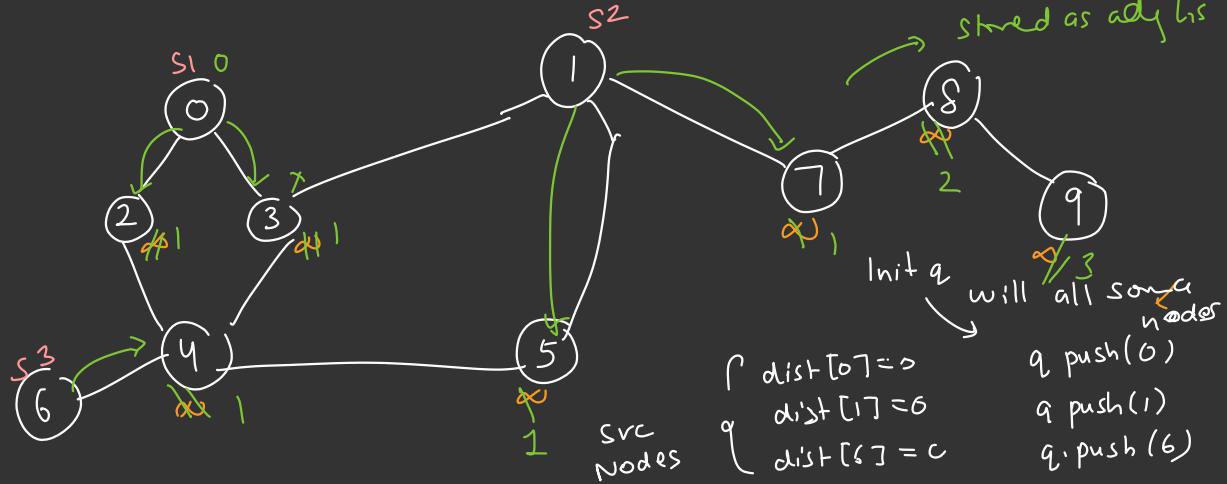
.

for all nodes starting any
of the source nodes

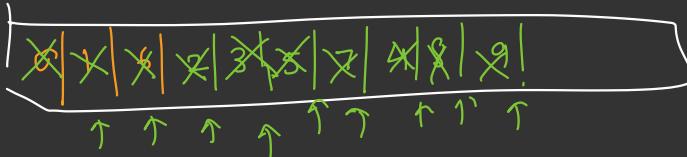


Unweighted graph
($\text{BI}=\mathcal{S}$)

$\tau=1$
 $\tau=2$
 $\tau=3$



$dist[n] = \infty$
 \downarrow
 not visited.



Init q
 will all source nodes
 $q.push(0)$
 $q.push(1)$
 $q.push(6)$

$\left\{ \begin{array}{l} dist[0] = \infty \\ dist[1] = 0 \\ dist[6] = \infty \end{array} \right.$

$BFS \log n$
 $\frac{while ()}{for (-nbs -)}$

$=$

3

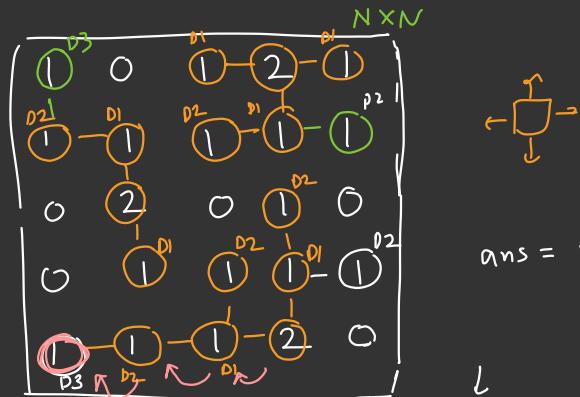
Rotten Oranges

1	0	1	2	1	
1	1	1	1	1	
0	2	0	1	0	
0	1	1	1	1	
1	1	1	2	0	

0 - empty
1 fresh orange
2 Rotten orange



Everyday any fresh orange adjacent to Rotten orange becomes rotten. \rightarrow 4 way
Find min days in which all oranges will become rotten.

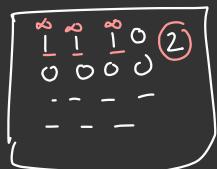


ans = max of all days

3

Take
the
longest
dist of FO

dist of all fresh
oranges
from nearest rotten
orange
[Multi-Source BFS]



Day = ∞

CODE int dist[m][n] = {∞, ∞, ...} → 0 if mat[i][j] = 0

queue <pair<int, int>> q;

// Init q with pos of Rotten Oranges

```
for(i=0 — M-1) {  
    for(j=0 — N-1) {  
        if (mat[i][j] == 2) {  
            q.push(pair(i,j))  
            dist[i,j] = 0  
        }  
    }  
}
```

// BFS - spread in 4 dir

each cell goes into q, atmost once

(2,3)

4 N²

= O(N²)

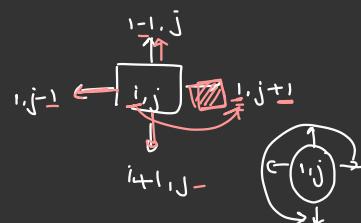
i,j = q.pop()

dx = {0, -1, 0, +1}

dy = {1, 0, -1, 0}

for (k=0, k<4; k++) {

ni = i+dx[k]
nj = j+dy[k]



ni, nj {
 i+0, j+1
 i-1, j
 i+0, j-1
}

not visited if orange is fresh

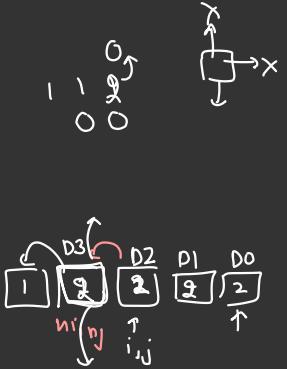
```

if (mat[ni][nj] == 0 or outside Sgrid)
    continue;
if (mat[ni][nj] == 1) {
    mat[ni][nj] = 2
    dist[ni][nj] = 1 + dist[i][j]
    q.push(pair(ni, nj))
}
}
}

```

if (mat[ni][nj] == 1) {
 mat[ni][nj] = 2
 dist[ni][nj] = 1 + dist[i][j]
 q.push(pair(ni, nj))
}

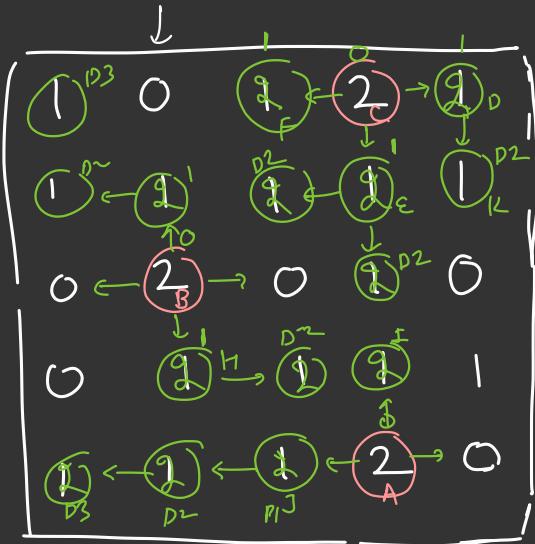
if (mat[ni][nj] == 0 or outside Sgrid)
 continue;



```

for (d, dist) {
    max_dist = Track the max val of d
    s
    return max_dist
}

```



q X X X X X F G H I J K
 ↑ ↑ ↑ ↑ ↑ ↑

⇒
$$\begin{bmatrix} 3 & 0 & 1 & 0 & 1 \\ 2 & 1 & 2 & 1 & 2 \\ - & - & - & - & - \\ 3 & - & - & - & - \end{bmatrix}$$

↑
 longest in dist mat ~~00000~~